

Чернівецький національний університет імені Юрія Федьковича
Навчально-науковий інститут фізико-технічних та
комп'ютерних наук
Кафедра комп'ютерних наук

ЗВІТ З ОБЧИСЛЮВАЛЬНОЇ ПРАКТИКИ
НА ТЕМУ:
«Розробка застосунку для страхової компанії»

Студента 1 курсу, групи 144Б
Спеціальність 122 “Комп'ютерні науки”
Приутеса А.Е.

(підпис)

Керівник доцент кафедри комп'ютерних наук

кандидат фізико-математичних наук
Довгунь А.Я.

(посада, вчене звання, науковий ступінь, прізвище
та ініціали)

Національна шкала: _____

Кількість балів: _____ Оцінка: ECTS _____

Чернівці – 2025

Вступ.....	3
1 Аналіз предметної області	4
1.1 Опис предметної області	4
1.2. Технічне завдання на розробку програмного продукту	7
1.3 Опис інструментів розробки	9
2 Розробка програмного продукту	12
2.1 Проектування бази даних.....	12
2.2 Створення прототипу інтерфейсу	17
2.3 Реалізація застосунку.....	20
2.4 Інструкція користувача.....	27
Висновок	32
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	32
Додаток А.....	33

ВСТУП

Метою створення застосунку для автоматизації роботи страхової компанії була потреба ефективного контролю за основними аспектами її діяльності, зокрема такими, як облік договорів страхування, клієнтів, страхових випадків та виплат. Впровадження програмного забезпечення дозволяє зменшити кількість рутинних операцій, прискорити обробку інформації та забезпечити її збереження.

Актуальність теми дослідження полягає в тому, що «Розробка застосунку для страхової компанії» дозволить значно зменшити витрати часу на виконання типових завдань, підвищити точність ведення обліку та загалом оптимізувати бізнес-процеси компанії.

Програмне забезпечення, яке розробляється для страхової компанії, повинно забезпечувати надійний облік страхових договорів, можливість подання страхових випадків, збереження даних про клієнтів та адміністрування доступу до системи. Важливо, щоб інтерфейс користувача був зручним, а робота з базою даних — швидкою і безпечною.

Для досягнення цієї мети були поставлені наступні задачі:

- виконати опис предметної області;
- розробити технічне завдання на створення програмного продукту;
- спроектувати базу даних, що містить інформацію про клієнтів договори, страхові випадки та виплати;
- створити інтерфейс користувача для зручної взаємодії з базою даних;
- реалізувати функціонал програми в середовищі Embarcadero C++ Builder;
- скласти інструкцію користувача для роботи із застосунком;

Предметом дослідження є автоматизація обліку клієнтів, договорів, страхових випадків та виплат у страховій сфері.

Звіт з обчислювальної практики складається зі вступу, основної частини (двох розділів), висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Страхова компанія – це фінансова установа, створена у формі акціонерного, повного, командитного товариства або товариства з додатковою відповідальністю з урахуванням особливостей, передбачених Законом України «Про страхування», що одержала у встановленому порядку ліцензію на здійснення страхової діяльності[4]. Предметна область охоплює сукупність процесів, пов'язаних із прийняттям страхових ризиків, укладанням договорів, обліком страхових випадків, нарахуванням та здійсненням страхових виплат.

Основна мета страхової компанії полягає в забезпеченні фінансового захисту клієнтів від можливих збитків, пов'язаних із ризиками, які можуть виникнути у повсякденному житті або внаслідок господарської діяльності.

Діяльність страхової компанії також передбачає взаємодію з агентами, які здійснюють пошук клієнтів та продаж страхових продуктів, а також з державними регуляторами, які контролюють дотримання чинного законодавства у сфері страхування.

Основний напрямок діяльності є надання страхових послуг фізичним та юридичним особам шляхом укладання договору з метою захисту особи від ризику. Компанія бере на себе зобов'язання виплатити страхове відшкодування або страхову суму при настанні страхового випадку, а клієнт зобов'язується сплачувати страхові внески відповідно до умов договору.

Основний напрям включає в себе: розробку та реалізації страховий продуктів, оцінку страхових ризиків, укладання та обслуговування страхових договорів, ведення обліку клієнтів та полісів, прийом заяв про страхові випадки, прийняття рішень щодо виплат страхового відшкодування.

Додаткові напрями діяльності включають в себе: інвестиційну діяльність, перестраховування, інформаційно-консультаційні послуги, агентська діяльність, освітня та соціальна діяльність.

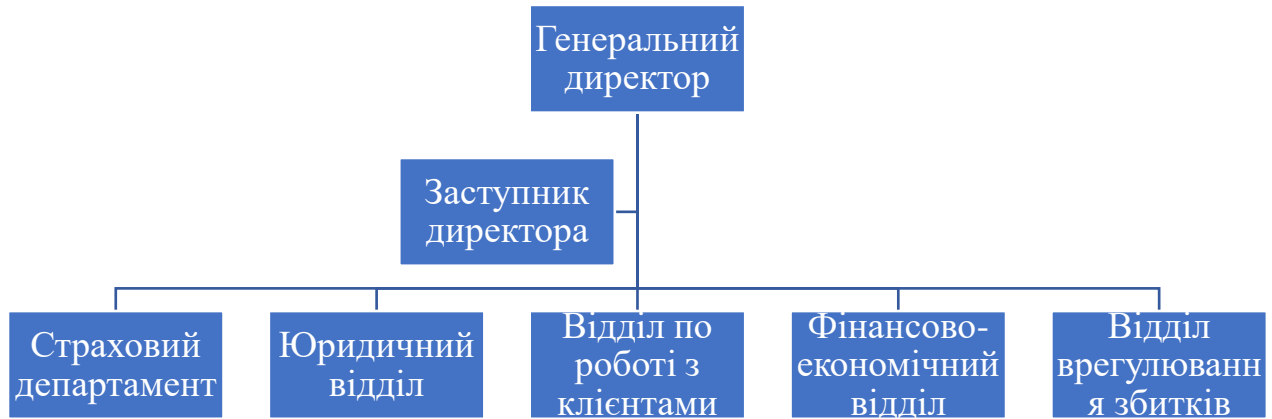


Рис. 1.1 Організаційна структура компанії

Генеральний директор: несе відповідальність за загальне стратегічне управління підприємством, приймає ключові управлінські рішення, затверджує плани розвитку, контролює фінансові показники та представляє компанію у взаємодії з зовнішніми організаціями — державними органами, партнерами, інвесторами.

Заступник директора: курує окремими напрямками діяльності, зокрема страхування, фінанси, правові питання, інформаційні технології, маркетинг та інші.

Страховий департамент: відповідає за розробку страхових продуктів, оцінку ризиків, підготовку договорів та загальну організацію процесу страхування. У його складі можуть діяти окремі сектори за напрямками: страхування життя, майна, здоров'я, автомобілів тощо. Працівники цього департаменту проводять розрахунок страхових тарифів, аналізують ринок страхових послуг, формують умови страхування відповідно до чинного законодавства.

Юридичний відділ: забезпечує правовий супровід діяльності компанії. Його працівники перевіряють правильність укладення договорів, проводять юридичну експертизу внутрішніх документів, консультують інших працівників з правових питань, супроводжують судові справи, у яких фігурує компанія, а також слідкують за відповідністю роботи страхової організації чинному законодавству України.

Відділ по роботі з клієнтами: Його завданням є консультування потенційних і діючих клієнтів, супровід під час оформлення договорів, надання допомоги при продовженні чи зміні полісів, а також розгляд звернень, скарг і запитів. Робота цього відділу безпосередньо впливає на репутацію компанії та рівень довіри з боку клієнтів.

Фінансово-економічний відділ: займається всіма аспектами бухгалтерського та фінансового обліку в компанії. Він контролює рух коштів, веде облік сплачених страхових внесків і здійснених виплат, формує фінансову звітність, складає бюджети та забезпечує своєчасне виконання зобов'язань перед клієнтами, державними органами і партнерами. Крім того, цей підрозділ відповідає за створення та облік страхових резервів, що є критично важливими для стабільності компанії.

Відділ врегулювання збитків: виконує розгляд страхових випадків. Саме сюди надходять заяви клієнтів про настання страхових подій. Спеціалісти цього відділу здійснюють перевірку фактів, проводять оцінку збитків (іноді спільно з незалежними експертами), приймають рішення про доцільність та розмір виплат, а також готують документи для виплати страхового відшкодування. У випадку порушення умов договору або недостовірності заявлених даних, відділ готує обґрунтовану відмову у виплаті.

Темою для проходження практики є розробка застосунку для надання послуг страхування компанії «Save».

1.2. Технічне завдання на розробку програмного продукту

1. Загальні відомості.

Інформаційна система для надання страхових послуг призначена для автоматизації діяльності страхової компанії. Система покликана полегшити процеси обслуговування клієнтів, укладення договорів, обробки страхових випадків, здійснення виплат та формування звітності.

2. Цілі та призначення.

Застосунок має досягти таких цілей:

- зменшення витрат часу на обробку інформації;
- централізоване зберігання даних у безпечному середовищі з архітектурою «клієнт-сервер»;
- можливість швидкого доступу до даних клієнтів, договорів, платежів, страхових випадків;
- зручне управління користувачами через систему ролей (клієнт, адміністратор);
- розширені можливості формування звітів;
- адміністрування користувачів і бази даних.

3. Характеристика об'єкта автоматизації.

Об'єктом автоматизації є страхова компанія, що надає послуги з добровільного та обов'язкового страхування (життя, майна, авто, здоров'я тощо).

4. Вимоги до застосунку.

Система повинна забезпечувати такі функціональні можливості:

- авторизація користувачів та контроль доступу за ролями ;
- реєстрація клієнтів та зберігання інформації про них;
- укладання та супровід договорів страхування;
- реєстрація страхових випадків і облік виплат;
- ведення платежів за договорами;

- формування звітів;
- безпечне зберігання даних;

5. Перелік робіт зі створення застосунку.

Розглянемо перелік робіт, спрямованих на створення застосунку для страхової компанії.

- розробка технічного завдання;
- проектування логічної моделі бази даних;
- створення прототипу інтерфейсу користувача;
- реалізація програмної логіки застосунку;
- тестування функціоналу на тестових даних;
- підготовка інструкції користувача.

6. Порядок розробки застосунку.

1. На основі технічного завдання виконується проектування бази даних.
2. Далі створюється макет інтерфейсу.
3. Реалізується прикладна логіка (введення/редагування клієнтів, договорів, виплат).
4. Додаються допоміжні модулі — формування звітів, захист доступу, журнал операцій.
5. Здійснюється повне тестування системи.

7. Порядок контролю та приймання застосунку.

Контроль та приймання застосунку виконується комісією у складі представника керівництва страхової компанії, IT-фахівця компанії(або простого користувача) та розробника.

Вимоги до введення застосунку в дію.

Для повноцінного запуску ІС потрібно:

- провести навчання персоналу роботи з ІС;
- забезпечити доступ до локальної бази даних;
- протестувати систему в реальному середовищі з обмеженим обсягом даних;

- за потреби коригування на основі результатів пробного запуску.

9.Вимоги до документування.

Супровідна документація повинна включати:

- опис предметної області (страхова компанія);
- схему структури бази даних;
- пояснення до архітектури застосунку;
- інструкції користувача;

10.Джерела розробки:

- завдання на проходження обчислювальної практики;
- технічне завдання;

11.Для того, щоб впровадити застосунок для класичних ігор у роботу, апаратне забезпечення повинно відповідати таким мінімальним нормам:

- процесор Intel Core i3 (1.4 GHz) або аналог
- Оперативна пам'ять 4 GB
- Жорсткий диск 250 GB

Програмне забезпечення:

- Операційна система: Windows 7 або новіша
- Платформа розробки: C++ Builder
- СУБД: Microsoft Access

1.3 Опис інструментів розробки

C++ Builder — це середовище швидкої розробки (RAD, Rapid Application Development), що дозволяє створювати настільні застосунки для Windows із використанням мови програмування C++. Розроблений компанією Embarcadero, цей інструмент поєднує у собі можливості візуального проектування інтерфейсу та потужного компілятора C++[2].

Середовище C++ Builder має наступні переваги:

- Підтримка візуального проектування форм (drag-and-drop інтерфейс).
- Висока швидкість створення прототипів.
- Інтеграція з базами даних, у тому числі Microsoft Access.
- Потужний компілятор C++, що дозволяє створювати швидкі й ефективні програми.
- Вбудовані компоненти для роботи з базами даних: TTable, TQuery, TDataSource, DBGrid тощо.
- Середовище C++ Builder має наступні недоліки:
- Обмежена підтримка сучасних бібліотек C++ та сторонніх фреймворків.
- Програми можуть мати більший розмір через статичне зв'язування з бібліотеками.
- Платна ліцензія.
- Менша популярність у порівнянні з Visual Studio або Qt.

Інтерфейс:

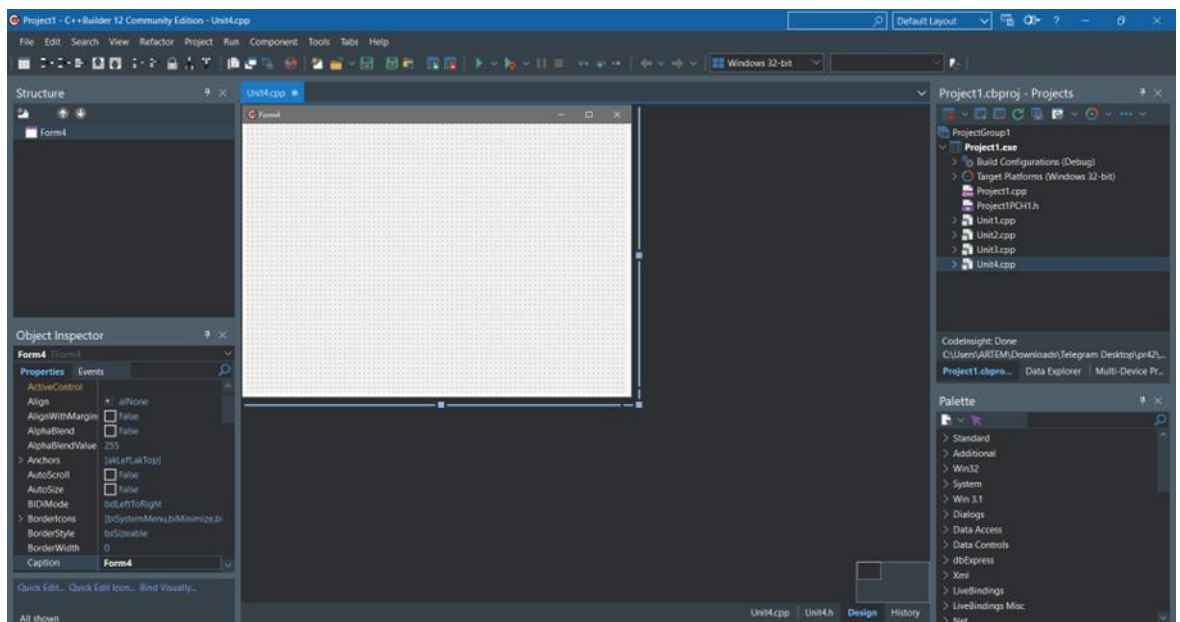


Рис. 1.2 Інтерфейс C++ Builder

Інтерфейс C++ Builder складається з таких основних частин:

- Form Designer - дозволяє візуально створювати вікна програми.
- Code Editor - для написання програмного коду на C++.
- Object Inspector - для налаштування властивостей об'єктів форми.
- Project Manager - для управління файлами проекту.
- Tool Palette - набір компонентів, які можна перетягувати на форму.

Microsoft Access— система управління базами даних, програма, що входить до складу пакету офісних програм Microsoft Office. Має широкий спектр функцій, включаючи зв'язані запити, сортування по різних полях, зв'язок із зовнішніми таблицями і базами даних.. Вона дозволяє створювати таблиці, запити, форми та звіти без потреби в написанні великої кількості SQL-коду, і добре підходить для невеликих або середніх проєктів[3].

Середовище Microsoft Access має наступні переваги:

- Проста у використанні навіть для користувачів без глибоких технічних знань.
- Інтеграція з іншими офісними програмами (Excel, Word).
- Швидке створення таблиць, запитів і звітів за допомогою вбудованих шаблонів.
- Можливість імпорту/експорту даних.
- Сумісність із C++ Builder через ODBC або драйвери Access Database Engine.

Середовище Microsoft Access має наступні недоліки:

- Обмежена масштабованість: підходить для невеликої кількості користувачів і записів .
- Відсутність підтримки складних транзакцій та тригерів, як у професійних СУБД (SQL Server, PostgreSQL).

- Відсутність кросплатформенності — працює лише в середовищі Windows.
- Невисока безпека, якщо не реалізувати додаткові механізми захисту.

Інтерфейс:

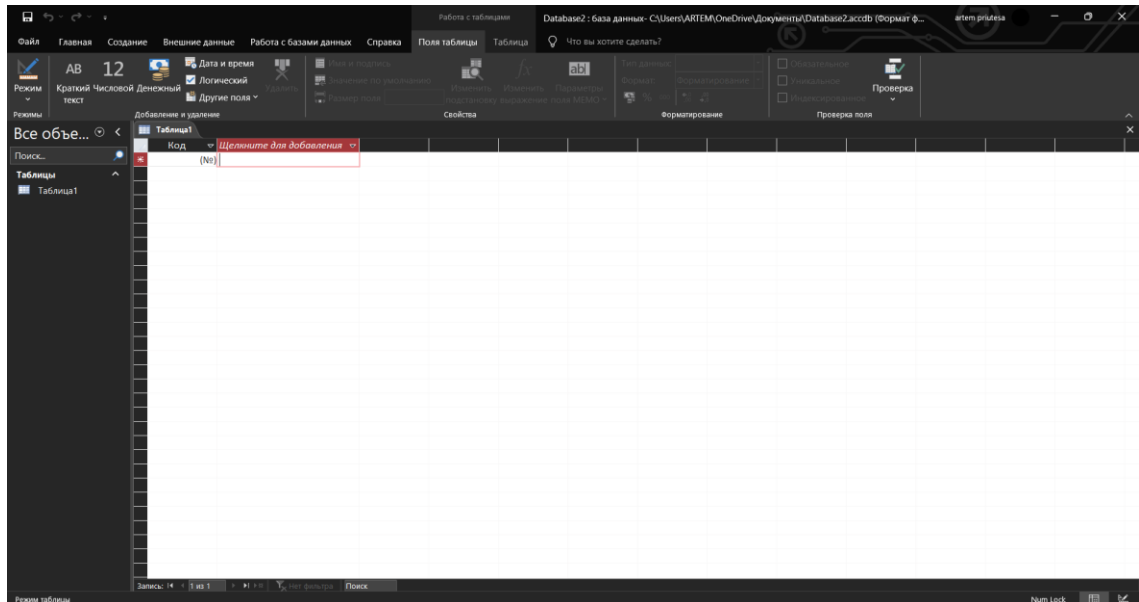


Рис. 1.3 Інтерфейс Microsoft Access

Інтерфейс Microsoft Access складається з:

- Таблиць — для зберігання даних.
- Запитів — для вибірки, фільтрації та обчислення даних.
- Форм — для візуального введення/редагування даних.
- Звітів — для генерації вихідної документації.
- Навігаційної панелі — доступ до всіх об'єктів бази.

2 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Проектування бази даних

Характеристика бази даних

У відповідності до проаналізованої предметної області можна виділити такі сутності:

- Користувач — це людина або компанія яка користується страховими послугами.

- Договір(поліс) - це документ між клієнтом і страховою компанією.
- Платіж – це запис про те, коли і скільки клієнт заплатив за страхування.
- Страховий випадок - фіксація подій, пов'язаних із договором страхування, при яких клієнт подає заяву на відшкодування.
- Страхова виплата – грошова сума яка надається у випадку підтвердження страхового випадку.

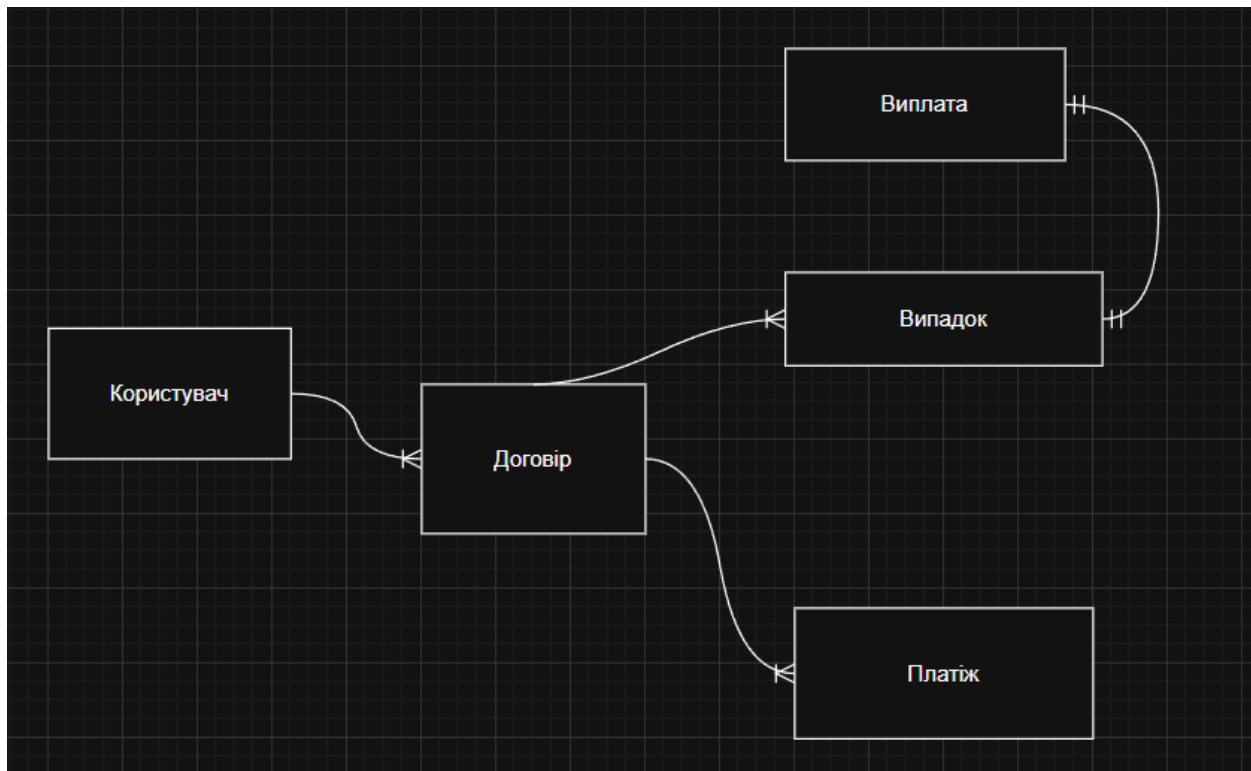


Рис.2.1 Концептуальна модель бази даних

Сутність Користувач використовує наступні реквізити:

- Код користувача(тип даних числовий ,ключове поле)
- Повне ім'я(тип даних текстовий)
- Логін(тип даних текстовий)
- Пароль(тип даних текстовий)
- Роль(тип даних текстовий)

Сутність Договір має наступні реквізити:

- Код договору(тип даних числовий, ключове поле)

- Код користувача(тип даних числовий, зв'язок з Користувач)
- Тип страхування(тип даних текстовий)
- Дата початку(тип даних числовий)
- Дата закінчення(тип даних числовий)
- Страхова сума(тип даних числовий)
- Статус договору(тип даних текстовий)

Сутність Страховий випадок має наступні реквізити:

- Код страхового випадку(тип даних числовий, ключове поле)
- Код договору(тип даних числовий, зв'язок з Договорі)
- Дата випадку(тип даних числовий)
- Коментар(тип даних текстовий)
- Статус випадку(тип даних текстовий)

Сутність Виплата має наступні реквізити:

- Код страхового випадку(тип даних числовий, ключове поле, зв'язок Страховим випадком)
- Код виплати(тип даних числовий)
- Дата виплати(тип даних числовий)
- Сума виплати(тип даних числовий)

Сутність Платіж має наступні реквізити:

- Код платежу(тип даних числовий, ключове поле)
- Код договору(тип даних числовий, зв'язок з договором)
- Дата платежу(тип даних числовий)
- Сума платежу(тип даних числовий)

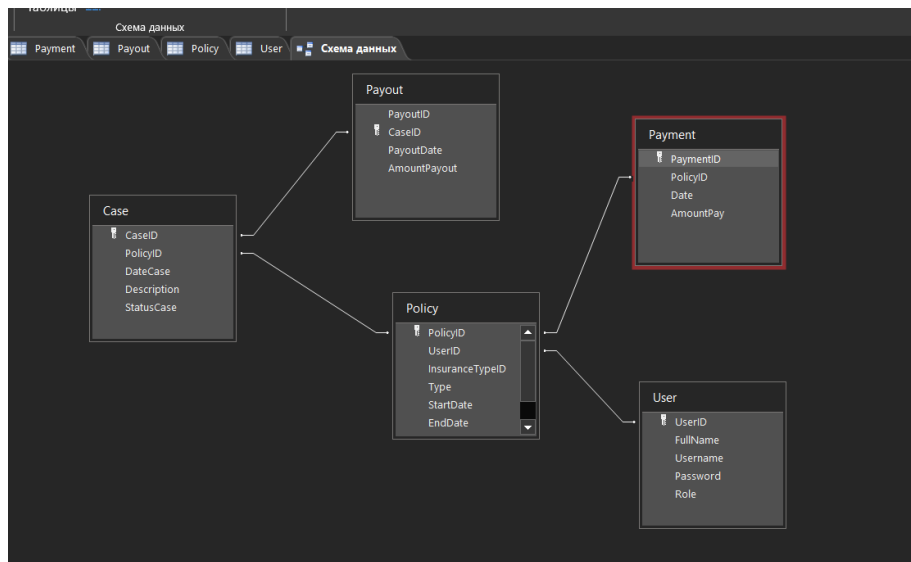


Рис.2.2 Схема бази даних

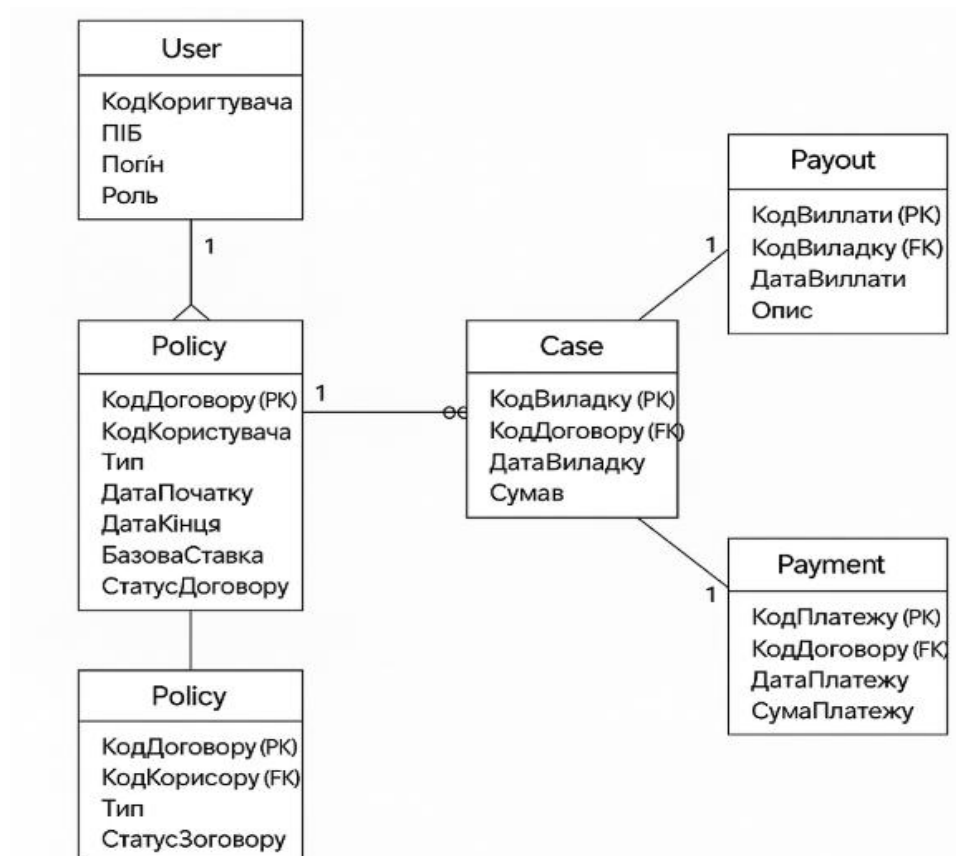


Рис.2.3 Логічна схема

Зв'язки між таблицями:

- Користувач 1—∞ Договір
- Договір 1—∞ Платіж

- Договір 1—∞ Випадок
- Випадок 1—1 Виплата

БД для страхової компанії створювалася в СУБД Access. Нижче вказані структури таблиць в режимі конструктора:

Имя поля	Тип данных
UserID	Счетчик
FullName	Короткий текст
Username	Короткий текст
Password	Короткий текст
Role	Короткий текст

Рис 2.4 Структура таблиці Користувач

Имя поля	Тип данных
PolicyID	Счетчик
UserID	Числовой
Type	Короткий текст
StartDate	Дата и время
EndDate	Дата и время
AmountPolise	Денежный
StatusPolisy	Короткий текст

Рис 2.5 Структура таблиці Договір

Имя поля	Тип данных
PayoutID	Счетчик
CaseID	Числовой
PayoutDate	Дата и время
AmountPayout	Денежный

Рис 2.6 Структура таблиці Страхова виплата

Case		Payment		Payout		Policy		User	
		Имя поля		Тип данных					
		PaymentID		Счетчик					
		PolicyID		Числовой					
		Date		Дата и время					
		AmountPay		Денежный					

Рис 2.7 Структура таблиці Платіж

Case		Payment		Payout		Policy		User	
		Имя поля		Тип данных					
		CaseID		Счетчик					
		PolicyID		Числовой					
		DateCase		Дата и время					
		Description		Короткий текст					
		StatusCase		Короткий текст					

Рис 2.8 Структура таблиці Страховий випадок

Перейдемо до створення прототипу інтерфейсу.

2.2 Створення прототипу інтерфейсу

При створенні своєї програми, я використовував такі компоненти C++ Builder'a: Label, Edit, Button, StringGrid, ComboBox, CheckBox, RadioButton, Timer, PopupMenu, MainMenu, Image, ProgressBar.

Label: Текстовий напис. Ключові параметри: **Caption** (текст), **Font** (шрифт), **Alignment** (вирівнювання).

Edit: Поле для введення або відображення тексту. Ключові параметри: **Text** (зміст), **Font** (шрифт), **ReadOnly** (заборона редагування).

Button: Кнопка, що викликає дію. Ключові параметри: **Caption** (текст), **Font** (шрифт), **OnClick** (подія натискання).

StringGrid: Таблиця для відображення і редагування даних. Ключові параметри: **ColCount** (кількість стовпців), **RowCount** (кількість рядків), **Cells** (доступ до даних комірок).

ComboBox: Випадаючий список. Ключові параметри: **Items** (елементи списку), **Text** (вибраний текст), **ItemIndex** (індекс вибраного елемента).

CheckBox: Прапорець для вибору опції (так/ні). Ключові параметри: **Caption** (текст), **Checked** (стан: встановлено/не встановлено).

RadioButton: Перемикач для вибору однієї опції з групи. Ключові параметри: **Caption** (текст), **Checked** (стан: вибрано/не вибрано), **GroupIndex** (групування перемикачів).

Timer: Компонент для виконання дій через задані інтервали часу. Ключові параметри: **Enabled** (активний/не активний), **Interval** (інтервал в мілісекундах).

PopupMenu: Контекстне меню, що з'являється при правому кліку. Ключовий параметр: **Items** (пункти меню).

MainMenu: Головне меню програми. Ключовий параметр: **Items** (пункти меню).

Image: Компонент для відображення зображень. Ключовий параметр: **Picture** (власне зображення).

ProgressBar: Графічний елемент, що відображає **хід виконання процесу** у вигляді заповненої смуги. Застосовується для візуалізації прогресу завантаження, обробки даних або довготривалих операцій. **Ключовий параметр:** **Position** (поточне значення прогресу).

DateTimePicker: Елемент для **зручного вибору дати і часу** через календар або часовий список. Дозволяє уникнути помилок при ручному введенні дат.

Ключовий параметр: **DateTime** (обране значення дати та/або часу).

Перейдемо до інтерфейсу основних форм застосунку.

Перша форма яка зустрічає користувача це форма завантаження на ній є елементи **ProgressBar**, **Timer** та **Label**.

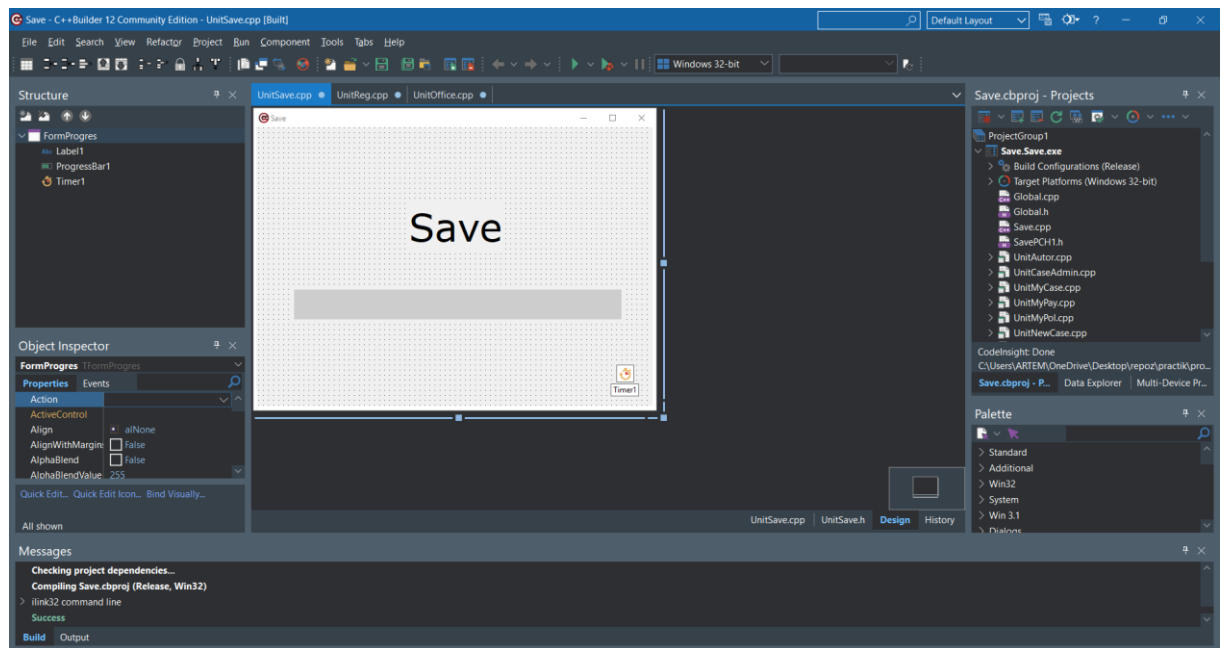


Рис. 2.9 Екран завантаження

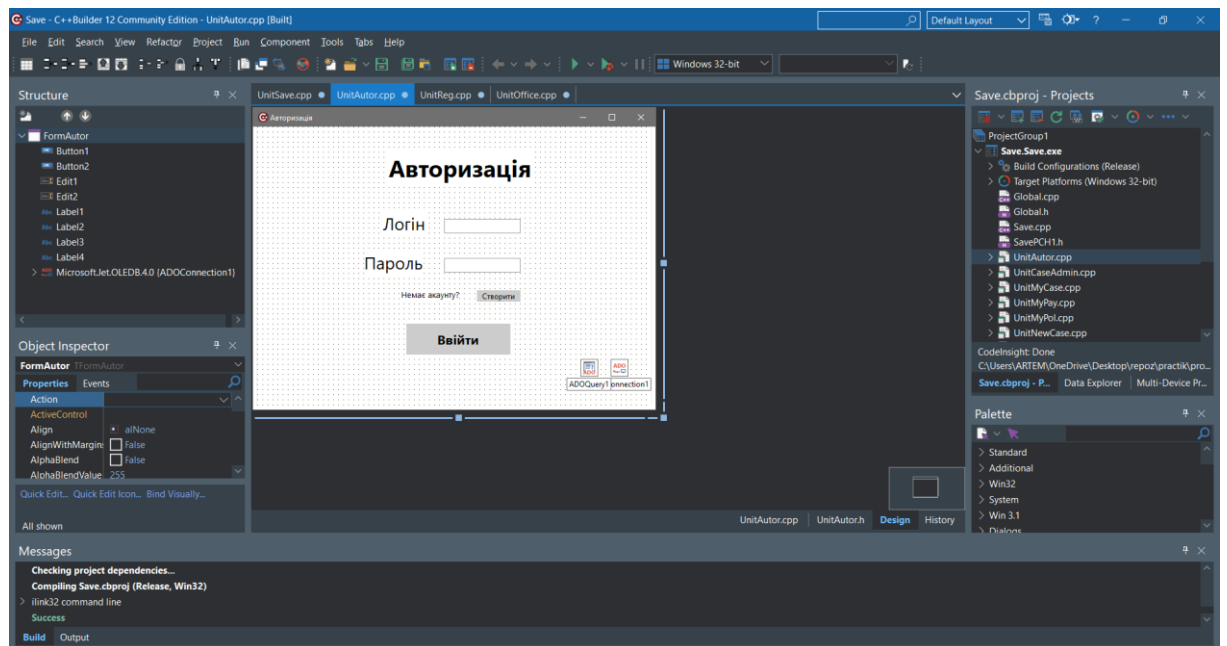


Рис. 2.10 Форма авторизації

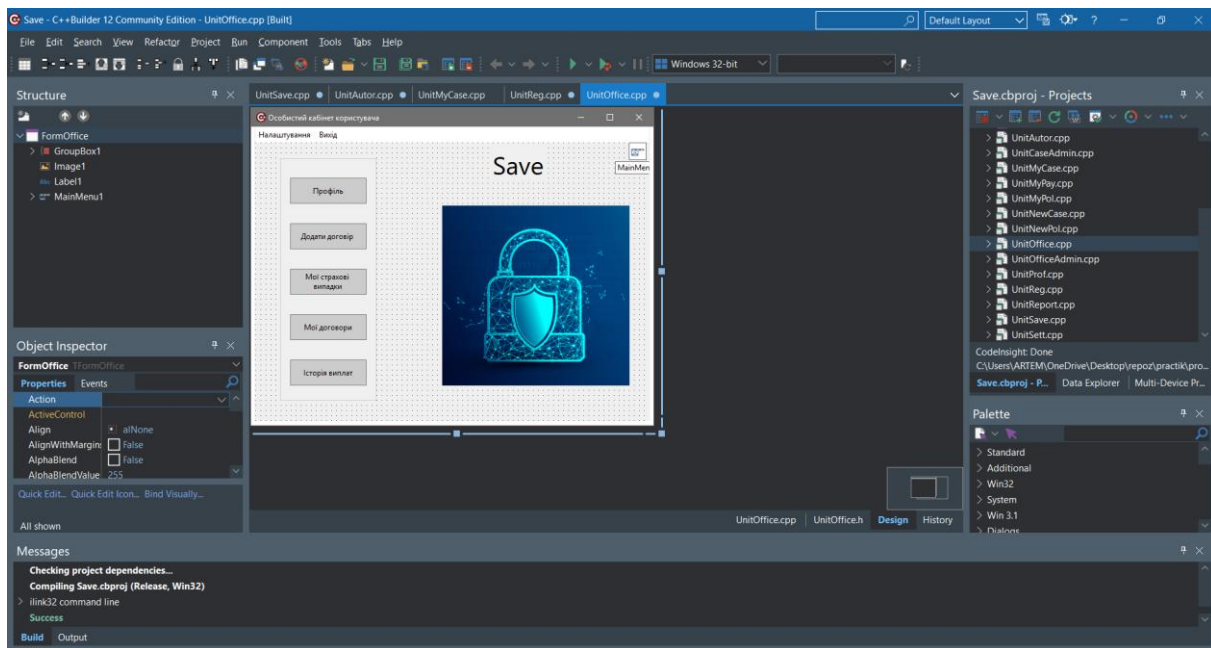


Рис. 2.11 Кабінет користувача

Вище показано інтерфейс основних форм застосунку в режимі розробника.

2.3 Реалізація застосунку

1. Підключення бази даних.

Програма використовує технологію ActiveX Data Objects (ADO) для встановлення зв'язку з базою даних та взаємодії з нею. ADO прикладний програмний інтерфейс для доступу до даних, розроблений компанією Microsoft (MS Access, MS SQL Server) і заснований на технології компонентів ActiveX. ADO дозволяє представляти дані з різноманітних джерел: (реляційної СУБД, текстових файлів тощо) в об'єктно орієнтованому програмуванні виді через драйвери OLE DB [4].

Для підключення до існуючої бази даних я використовував компоненти ADOConnection та ADOQuery для підключення бази даних до компоненту DBGrid я використовував DataSource.

На початку запуску програми компонент ADOConnection підключається до бази даних після успішного підключення компонент ADOQuery можна використовувати для SQL-операцій. Результати запитів (SELECT) через компонент DataSource відображаються у візуальних

компонентах, дозволяючи користувачеві переглядати та взаємодіяти з даними.

2. Авторизація та реєстрація користувачів.

У системі реалізовано повноцінний механізм авторизації та реєстрації користувачів, що забезпечує контроль доступу до програми залежно від ролі. Авторизація і реєстрація винесені в окремі форми TFormAutor та TFormReg, між якими можливий перехід.

Після запуску програми користувач потрапляє на форму авторизації, де має ввести логін і пароль. Якщо будь-яке з полів порожнє, система виводить повідомлення з проханням заповнити всі поля. Далі виконується SQL-запит до таблиці User, у якому перевіряється відповідність введених логіна та пароля існуючим даним у базі. При знаходженні відповідного запису зчитуються код користувача, ППП і роль. Залежно від ролі відкривається відповідна форма: для адміністратора — форма TFormOfficeAdmin, для звичайного користувача — TFormOffice. Успішний вхід супроводжується вітальним повідомленням. У випадку неправильного логіна чи пароля виводиться повідомлення про помилку.

Якщо користувач ще не має облікового запису, він може натиснути кнопку переходу до реєстрації. Форма реєстрації дозволяє ввести ППП, логін, пароль та підтвердження пароля. Перед додаванням нового користувача до бази виконується перевірка: всі поля мають бути заповнені, паролі мають збігатися, пароль має містити щонайменше чотири символи. Додатково перевіряється унікальність логіна за допомогою окремої SQL-команди. Якщо логін уже зайнятий, виводиться відповідне повідомлення, а поле очищується для повторного введення.

У разі успішного проходження всіх перевірок дані нового користувача записуються до таблиці User, після чого виводиться повідомлення про

успішну реєстрацію, а користувач повертається до форми авторизації для входу в систему.

Обидві форми використовують ADO-компоненти для підключення до бази даних Access. При створенні форм ініціалізується з'єднання з БД, і в разі невдачі користувач отримує відповідне повідомлення, після чого програма завершується. При закритті форм відбувається правильне закриття запитів і з'єднання.

Для реалізації авторизації я використовував такий код:

```
void __fastcall TFormAutor::Button1Click(TObject *Sender)
{
    AnsiString username = Edit1->Text.Trim();
    AnsiString password = Edit2->Text;
    if (username.IsEmpty() || password.IsEmpty()) {
        ShowMessage("Будь ласка, введіть логін та пароль.");
        return;
    }
    try
    {
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }
        ADOQuery1->SQL->Clear();
        ADOQuery1->SQL->Add("SELECT * FROM [User] WHERE Логін = :Логін
AND Пароль = :Пароль");
        ADOQuery1->Parameters->ParamByName("Логін")->Value = username;
        ADOQuery1->Parameters->ParamByName("Пароль")->Value = password;
        ADOQuery1->Open();
        if (!ADOQuery1->Eof) {
            UserID      = ADOQuery1->FieldByName("КодКористувача")-
>AsInteger;
            Username = ADOQuery1->FieldByName("ПІП")->AsString;
            AnsiString role = ADOQuery1->FieldByName("Роль")-
>AsString.ToLowerCase();
            ShowMessage("Ласкаво просимо, " + Username + "!");
            if (role == "адміністратор") {
                FormOfficeAdmin = new TFormOfficeAdmin(this);
                FormOfficeAdmin->Show();
            } else {
                FormOffice = new TFormOffice(this);
                FormOffice->Show();
            }
            Hide();
        } else {
            ShowMessage("Невірний логін або пароль.");
        }
    }
    catch (Exception &E)
    {
        ShowMessage("Сталася помилка: " + E.Message);
    }
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
}
```

```

    }
}
}
Для реалізації реєстрації я використовував такий програмний код:
void __fastcall TFormReg::Button1Click(TObject *Sender)
{
    AnsiString pip = EditPIP->Text.Trim();
    AnsiString login = EditLog->Text.Trim();
    AnsiString password = EditPas->Text;
    AnsiString confirmPassword = EditPas2->Text;
    AnsiString role = "клієнт";
    if (pip.IsEmpty() || login.IsEmpty() || password.IsEmpty() ||
confirmPassword.IsEmpty()) {
        ShowMessage("Будь ласка, заповніть всі обов'язкові поля .");
        return;
    }
    if (password != confirmPassword) {
        ShowMessage("Паролі не співпадають.");
        EditPas->Clear();
        EditPas2->Clear();
        EditPas->SetFocus();
        return;
    }
    if (password.Length() < 4) {
        ShowMessage("Пароль має бути не менше 4 символів.");
        return;
    }
    if (IsLoginExists(login)) {
        ShowMessage("Користувач з таким логіном вже існує.");
        EditLog->Clear();
        EditLog->SetFocus();
        return;
    }
    try
    {
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }

        ADOQuery1->SQL->Clear();
        ADOQuery1->SQL->Add("INSERT INTO [User] (ПІП, Логін, Пароль,
Роль) VALUES (:PIP, :Login, :Password, :Role)");

        ADOQuery1->Parameters->ParamByName("PIP")->Value = pip;
        ADOQuery1->Parameters->ParamByName("Login")->Value = login;
        ADOQuery1->Parameters->ParamByName("Password")->Value =
password;

        ADOQuery1->Parameters->ParamByName("Role")->Value = role;

        ADOQuery1->ExecSQL();

        ShowMessage("Реєстрація успішна! Ласкаво просимо, " + login + "!");

        ModalResult = mrOk;
    }
    catch (Exception &E)
    {
        ShowMessage("Помилка під час реєстрації: " + E.Message);
    }
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
}

```

```

    }
}
Додатково функція для перевірки логіна:
bool TFormReg::IsLoginExists(const AnsiString& login)
{
    bool exists = false;
    if (!ADOConnection1->Connected) {
        ShowMessage("Помилка: З'єднання з базою даних відсутнє.");
        return true;
    }
    try
    {
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }
        ADOQuery1->SQL->Clear();
        ADOQuery1->SQL->Add("SELECT COUNT(*) FROM [User] WHERE Логін =
:Логін");

        ADOQuery1->Parameters->ParamByName("Логін")->Value = login;
        ADOQuery1->Open();
        if (!ADOQuery1->Eof) {
            if (ADOQuery1->Fields->Fields[0]->AsInteger > 0) {
                exists = true;
            }
        }
    }
    catch (Exception &E)
    {
        ShowMessage("Помилка перевірки логіна: " + E.Message);
        exists = true;
    }
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
    return exists;
}

```

Основний функціонал.

Після авторизації користувач переходить на форму Кабінет користувача на якій є основний функціонал застосунку

До основного функціоналу користувача можна виділити:

- Створення нових договорів;
- Перегляд наявних;
- Подання страхових випадків;
- Перегляд страхових випадків;
- Перегляд історії страхових виплат;

- Налаштування стилів застосунку;

Якщо користувач зайшов під логіном адміністратора то він переходить на форму Офіс адміністратора в якій є функціонал:

- Перегляд та зміна статусу поданих страхових випадків;
- Створення звітів про користувачів;

Додатковий функціонал

Система налаштувань тем у застосунку дозволяє користувачу змінювати зовнішній вигляд інтерфейсу за допомогою вибору однієї з доступних тем — наприклад, світлої, темної або нічної. Цей вибір виконується на спеціальній формі "Налаштування", де користувач може обрати одну з тем за допомогою GroupBox, що містить три RadioButton. Кожна з тем відповідає певному стилю візуального оформлення програми. Наприклад, світла тема — це стандартне оформлення Windows, темна — сучасний темний стиль інтерфейсу, а нічна — ще темніший стиль, який зменшує навантаження на очі вночі.

Коли користувач натискає кнопку "Застосувати", програма зберігає вибраний стиль у файл налаштувань. Це може бути .ini або .txt файл, наприклад settings.ini, у який записується значення обраної теми. Для цього використовується клас TIniFile, який дозволяє читати і записувати прості конфігураційні дані.

Під час наступного запуску застосунку, у методі FormCreate головної форми або перед її показом, цей файл читається. Програма зчитує збережене значення і автоматично встановлює відповідний стиль, використовуючи Application->StyleName або TStyleManager::SetStyle. Таким чином, користувачу не потрібно щоразу вручну вибирати тему — програма сама запам'ятовує його вибір.

Цей підхід забезпечує просту і зручну персоналізацію інтерфейсу, зберігаючи комфорт користувача навіть після перезапуску програми.

Нище показаний код для завантаження та зберігання стилів між запусками застосунку

```
void LoadSettings() {
    TIniFile* ini = new TIniFile(ExtractFilePath(Application-
>ExeName) + "Settings.ini");
    CurrentTheme = ini->ReadString("Appearance", "Theme", "Light");
    delete ini;
}
void SaveSettings() {
    TIniFile* ini = new TIniFile(ExtractFilePath(Application-
>ExeName) + "Settings.ini");
    ini->WriteString("Appearance", "Theme", CurrentTheme);
    delete ini;
}
void ApplyStyle() {
    if (CurrentTheme == "Dark")
        TStyleManager::TrySetStyle("Carbon");
    else if (CurrentTheme == "Night")
        TStyleManager::TrySetStyle("Tablet Dark");
    else
        TStyleManager::TrySetStyle("Windows10");
}
```

Код для кнопки застосувати на формі налаштування:

```
if (RadioLight->Checked)
    CurrentTheme = "Light";
else if (RadioDark->Checked)
    CurrentTheme = "Dark";
else if (RadioNight->Checked)
    CurrentTheme = "Night";

SaveSettings();
ApplyStyle();

for (int i = 0; i < Screen->FormCount; i++) {
    Screen->Forms[i]->Refresh();
}
ShellExecute(NULL, L"open", Application->ExeName.w_str(), NULL, NULL,
SW_SHOWNORMAL);
Application->Terminate();
```

2.4 Інструкція користувача

Після запуску застосунку користувач побачить екран завантаження коли прогрес дійде до кінця з'явиться форма авторизації (Рис 2.13) при наявності акаунту потрібно ввести дані користувача логін та пароль.

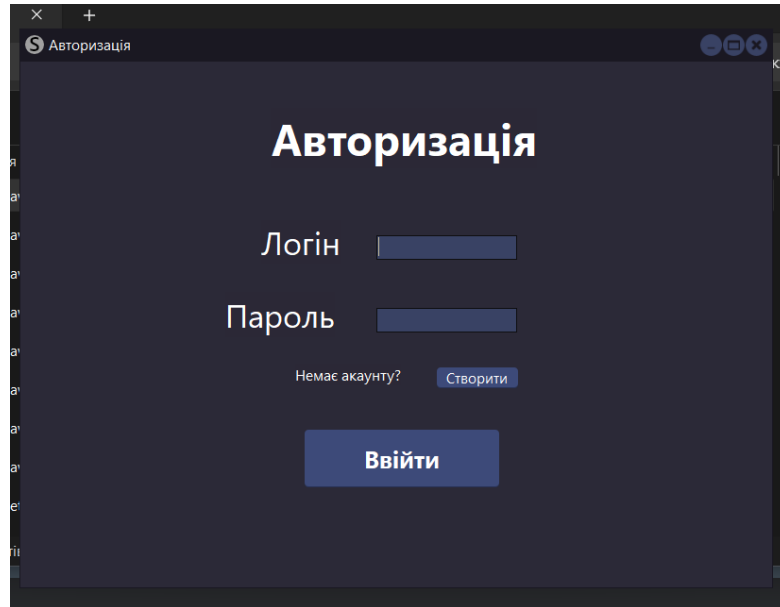
The image shows a web browser window with the title 'Авторизація' (Authorization). The main heading is 'Авторизація' in a large, bold font. Below it, there are two input fields: 'Логін' (Login) and 'Пароль' (Password). Under the password field, there is a link 'Немає акаунту?' (No account?) and a button 'Створити' (Create). At the bottom, there is a large blue button labeled 'Ввійти' (Login).

Рис. 2.13 Форма авторизації

Якщо акаунту немає потрібно натиснути на кнопку створити після цього користувача направить на форму реєстрації(Рис.2.14) де потрібно ввести свої дані після введення потрібно повернутися на форму авторизації і ввести дані логін і пароль.

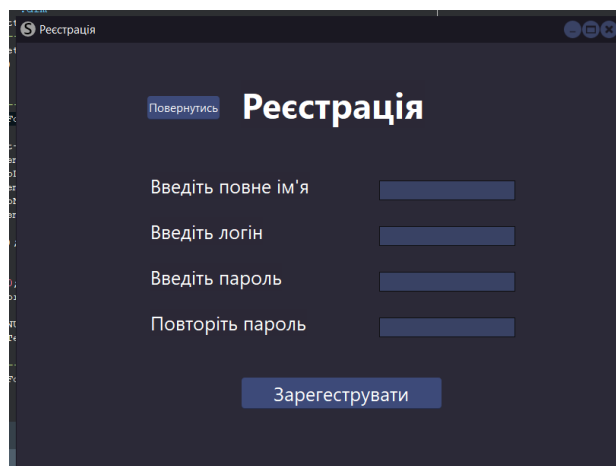
The image shows a web browser window with the title 'Реєстрація' (Registration). There is a link 'Повернутись' (Back) in the top left. The main heading is 'Реєстрація' in a large, bold font. Below it, there are four input fields: 'Введіть повне ім'я' (Enter full name), 'Введіть логін' (Enter login), 'Введіть пароль' (Enter password), and 'Повторіть пароль' (Repeat password). At the bottom, there is a large blue button labeled 'Зареєструвати' (Register).

Рис. 2.14 Форма реєстрації

Після успішної авторизації користувача направить на форму Офіс(Рис. 2.15) користувача на якій зосереджений функціонал застосунку.

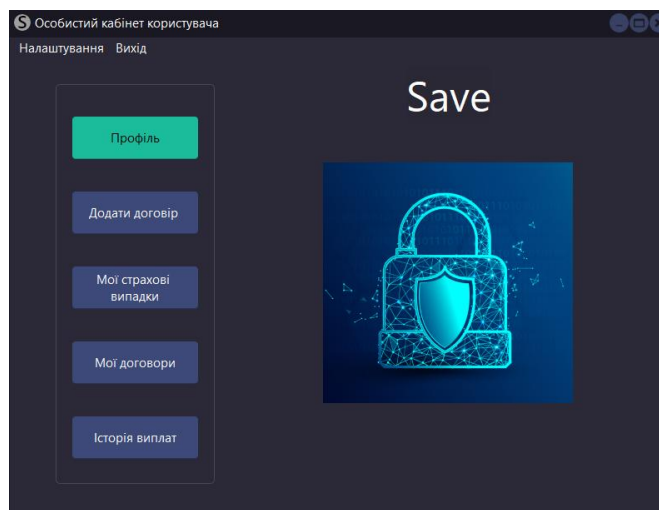


Рис 2.15 Кабінет користувача

Перший пункт це Профіль при натисканні на нього створиться форма профіль(Рис. 2.16) де є інформація про даного користувача.

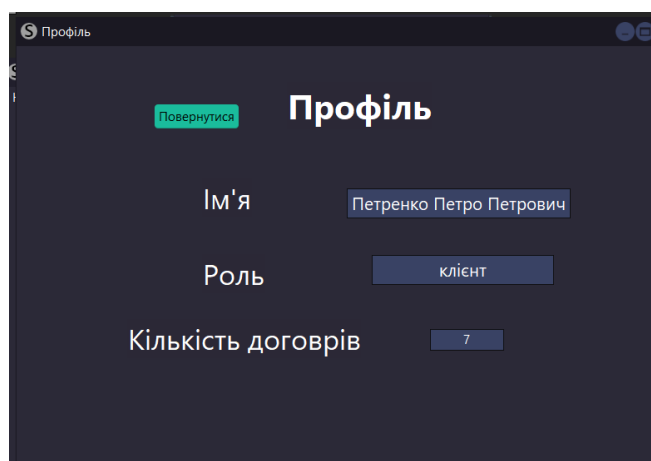


Рис. 2.16 Профіль користувача

Натиснувши кнопку Додати договір створиться форма Додати договір(Рис 2.17) в якій користувач може вибрати тип договору(автострахування або страхування життя), дату початку вказати термі дії договору, користувачу покажеться в полі щорічна оплата сума яку він щорічно має сплачувати для підтримки договору. Після натискання на кнопку Створити створюється перший платіж і в Пункт Мої договори(Рис 2.18) додається цей договір. Якщо потрібно відмінити створення натискаємо на кнопку Скасувати.

Рис.2.17 Форма створення договору

КодДоговору	КодКористувач	Тип	ДатаПочатку	ДатаКіця	БазоваСтавка	СтатусДоговору
1		автострахування	01.06.2025	30.06.2030	61000	триває
2		страхування жи	01.05.2025	30.06.2017	30000	триває
10		Автострахуванн	25.06.2025	25.06.2026	1000	Розглядається
11		Автострахуванн	30.06.2025	30.06.2031	6000	Розглядається
15		Автострахуванн	27.06.2025	27.06.2038	13000	Розглядається
16		Страхування жи	01.05.2025	01.05.2028	6000	Розглядається
17		Страхування жи	29.06.2025	29.06.2029	2000	Розглядається

Рис.2.18 Форма Мої договори

Якщо потрібно подати страховий випадок переходимо на форму Мої страхові випадки натиснувши на відповідну кнопку тут можна переглянути страхові випадки користувача і до якого договору вони приписані. При потребі подати страховий випадок нажимаємо на відповідну кнопку для користувача відкриється форма Подати страховий випадок в якій користувач вказує коли стався випадок, до якого договору буде приписаний страховий випадок та короткий опис випадку.

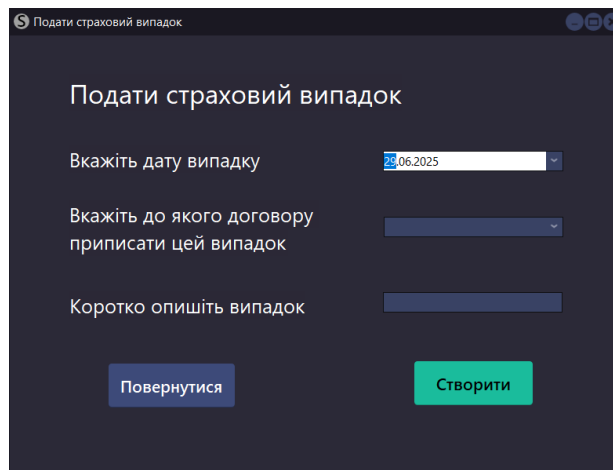


Рис 2.19 Форма Подання випадку

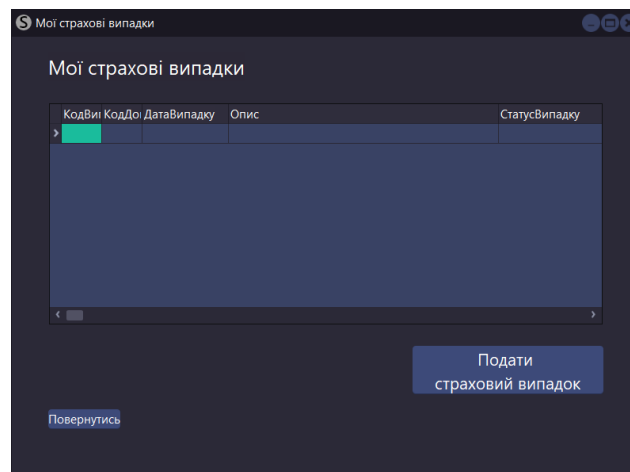


Рис 2.20 Форма Мої випадки

В останньому пункті Історія виплат можна переглянути історію виплат для страхових випадків статус яких був змінений адміністратором на схвалено.

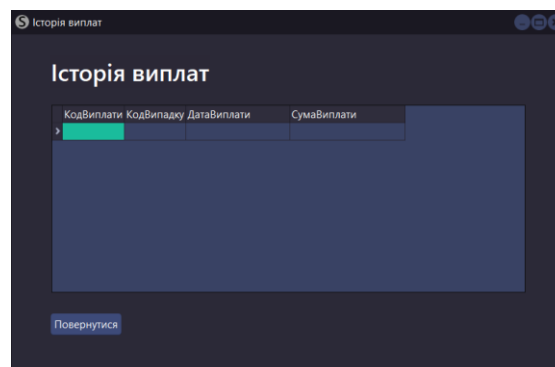


Рис 2.21 Форма Історія виплат

У випадку якщо користувач адміністратор він повинен при авторизації ввести логін і пароль я були уже створені. При успішній

авторизації адміністратора посилає на форму Офіс адміністратора в якому є пункти Переглянути страхові випадки, Звіт.

При натисканні на кнопку Переглянути подані заяви перед адміністратором з'явиться форма на якій є таблиця з всіма поданими випадками, кнопкою змінити статус. При потребі змінити статус адміністратор вибирає для якого випадку він хоче натиснувши на потрібний випадок в таблиці, після цього вибрати на який статус потрібно змінити в кінці натиснути кнопку Змінити статус при успішному змініні виведеться повідомлення про успішну зміну.

При виборі пункту Звіт адміністратору покажиться форма створення звіту в якому є поле для введення повного ім'я користувача про якого він хоче отримати звіт про його дані і кнопка Створити звіт після натискання відкриється меню вибору куди створиться звіт.

При потребі змінити тему застосунку незалежно від того який користувач потрібно у верхньому меню вибрати потрібну вкладку Налаштування після цього користувачу відкриється меню налаштувань в якому є вибір теми при виборі потрібної треба натиснути кнопку застосувати після цього застосунок перезавантажиться і користувачу знову потрібно буде ввести свій логін і пароль.

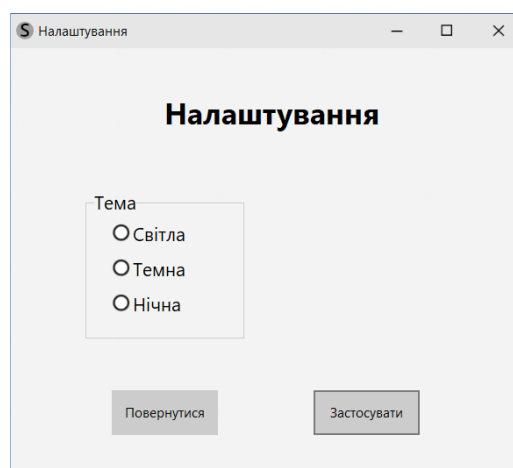


Рис 2.23 Форма Налаштувань

На цьому інструкція користувача закінчена.

ВИСНОВОК

Розроблений застосунок для надання страхових послуг значно спрощує надання послуг. Завдяки зручному графічному інтерфейсу користувач може швидко оформити страховий договір та швидко і просто подати страховий випадок.

Програма забезпечує надійне збереження інформації у базі даних, дозволяє формувати звіти. Це робить її корисним інструментом для користувачів.

При проходженні практики були виконані такі задачі:

- виконати опис предметної області;
- розробити технічне завдання на створення програмного продукту;
- спроектувати базу даних, що містить інформацію про клієнтів договори, страхові випадки та виплати;
- створити інтерфейс користувача для зручної взаємодії з базою даних;
- реалізувати функціонал програми в середовищі Embarcadero C++ Builder;
- скласти інструкцію користувача для роботи із застосунком;

Оскільки всі задачі виконані, можемо стверджувати, що дослідження своєї мети – досягло.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **ADO** / Wikipedia [Режим доступу]: URL: https://uk.wikipedia.org/wiki/ActiveX_Data_Objects (дата звернення: 24.06.2025).
2. **C++ Builder** / Wikipedia [Режим доступу]: URL: https://uk.wikipedia.org/wiki/C%2B%2B_Builder (дата звернення: 24.06.2025).

3. **Microsoft Access** / Wikipedia [Режим доступу]: URL:
https://uk.wikipedia.org/wiki/Microsoft_Access (дата звернення:
24.06.2025).
4. **Страхова компанія** / Consulting [Режим доступу]: URL:
<https://consulting.pp.ua/strahova-kompaniya> (дата звернення:
24.06.2025).

ДОДАТОК А

Лістинг програмного коду

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
#include <tchar.h>  
//-----  
#include <Vcl.Styles.hpp>  
#include <Vcl.Themes.hpp>  
USEFORM("UnitOfficeAdmin.cpp", FormOfficeAdmin);  
USEFORM("UnitOffice.cpp", FormOffice);  
USEFORM("UnitNewPol.cpp", FormNewPol);  
USEFORM("UnitNewCase.cpp", FormNewCase);  
USEFORM("UnitProf.cpp", FormProf);  
USEFORM("UnitSett.cpp", FormSett);  
USEFORM("UnitSave.cpp", FormProgres);  
USEFORM("UnitReport.cpp", FormReport);  
USEFORM("UnitReg.cpp", FormReg);  
USEFORM("UnitAutor.cpp", FormAutor);  
USEFORM("UnitMyPol.cpp", FormMyPol);  
USEFORM("UnitMyPay.cpp", FormMyPay);  
USEFORM("UnitMyCase.cpp", FormMyCase);  
USEFORM("UnitCaseAdmin.cpp", FormCasesAdmin);  
//-----  
int WINAPI _tWinMain(HINSTANCE, HINSTANCE, LPTSTR, int)  
{  
    try  
    {  
        Application->Initialize();  
        Application->MainFormOnTaskBar = true;  
        Application->CreateForm(__classid(TFormProgres), &FormProgres);  
        Application->CreateForm(__classid(TFormAutor), &FormAutor);  
        Application->CreateForm(__classid(TFormReg), &FormReg);  
        Application->CreateForm(__classid(TFormOffice), &FormOffice);  
        Application->CreateForm(__classid(TFormNewPol), &FormNewPol);  
        Application->CreateForm(__classid(TFormMyPol), &FormMyPol);  
        Application->CreateForm(__classid(TFormMyCase), &FormMyCase);  
        Application->CreateForm(__classid(TFormProf), &FormProf);  
        Application->CreateForm(__classid(TFormNewCase), &FormNewCase);  
    }  
}
```

```

        Application->CreateForm(__classid(TFormMyPay), &FormMyPay);
        Application->CreateForm(__classid(TFormSett), &FormSett);
        Application->CreateForm(__classid(TFormCasesAdmin), &FormCasesAdmin);
        Application->CreateForm(__classid(TFormOfficeAdmin),
&FormOfficeAdmin);
        Application->CreateForm(__classid(TFormReport), &FormReport);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}
//-----
//-----

#include <vcl.h>
#pragma hdrstop

#include "UnitAutor.h"
#include "UnitReg.h"
#include "UnitOffice.h"
#include "UnitOfficeAdmin.h"
#include "Global.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormAutor *FormAutor;
//-----
__fastcall TFormAutor::TFormAutor(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
void __fastcall TFormAutor::Button1Click(TObject *Sender)
{
    AnsiString username = Edit1->Text.Trim();
    AnsiString password = Edit2->Text;

    if (username.IsEmpty() || password.IsEmpty()) {
        ShowMessage("Будь ласка, введіть логін та пароль.");
        return;
    }
    try
    {
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }
    }

```

```

        ADOQuery1->SQL->Clear();
        ADOQuery1->SQL->Add("SELECT * FROM [User] WHERE Логін = :Логін AND
Пароль = :Пароль");
        ADOQuery1->Parameters->ParamByName("Логін")->Value = username;
        ADOQuery1->Parameters->ParamByName("Пароль")->Value = password;
        ADOQuery1->Open();
        if (!ADOQuery1->Eof) {
            UserID = ADOQuery1->FieldByName("КодКористувача")->AsInteger;
            Username = ADOQuery1->FieldByName("ПІП")->AsString;
            AnsiString role = ADOQuery1->FieldByName("Роль")-
>AsString.ToLowerCase();
            ShowMessage("Ласкаво просимо, " + Username + "!");
            if (role == "адміністратор") {
                FormOfficeAdmin = new TFormOfficeAdmin(this);
                FormOfficeAdmin->Show();
            } else {
                FormOffice = new TFormOffice(this);
                FormOffice->Show();
            }
            Hide();
        } else {
            ShowMessage("Невірний логін або пароль.");
        }
    }
    catch (Exception &E)
    {
        ShowMessage("Сталася помилка: " + E.Message);
    }
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
}
//-----
void __fastcall TFormAutor::FormCreate(TObject *Sender)
{
    ApplyStyle();
    try
    {
        ADOConnection1->Connected = true;
    }
    catch (Exception &E)
    {
        ShowMessage("Помилка підключення до БД: " + E.Message + "\nПеревірте
шлях до users_db.accdb та налаштування ADOConnection.");
        Application->Terminate();
    }
}
//-----
void __fastcall TFormAutor::FormClose(TObject *Sender, TCloseAction &Action)
{
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
    if (ADOConnection1->Connected) {
        ADOConnection1->Close();
    }
}
//-----
void __fastcall TFormAutor::Button2Click(TObject *Sender)
{
    FormReg->Show();
}

```

```

        Hide();
    }
    //-----
    //-----

#include <vcl.h>
#pragma hdrstop

#include "UnitReg.h"
#include "UnitAutor.h"
#include "UnitOffice.h"
#include "Global.h"

//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormReg *FormReg;
//-----
__fastcall TFormReg::TFormReg(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
bool TFormReg::IsLoginExists(const AnsiString& login)
{
    bool exists = false;
    if (!ADOConnection1->Connected) {
        ShowMessage("Помилка: З'єднання з базою даних відсутнє.");
        return true;
    }
    try
    {
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }
        ADOQuery1->SQL->Clear();
        ADOQuery1->SQL->Add("SELECT COUNT(*) FROM [User] WHERE Логін =
:Логін");
        ADOQuery1->Parameters->ParamByName("Логін")->Value = login;
        ADOQuery1->Open();
        if (!ADOQuery1->Eof) {
            if (ADOQuery1->Fields->Fields[0]->AsInteger > 0) {
                exists = true;
            }
        }
    }
    catch (Exception &E)
    {
        ShowMessage("Помилка перевірки логіна: " + E.Message);
        exists = true;
    }
    if (ADOQuery1->Active) {
        ADOQuery1->Close();
    }
    return exists;
}

void __fastcall TFormReg::Button1Click(TObject *Sender)
{
    AnsiString pip = EditPIP->Text.Trim();
    AnsiString login = EditLog->Text.Trim();

```

```

        AnsiString password = EditPas->Text;
        AnsiString confirmPassword = EditPas2->Text;
        AnsiString role = "клієнт";
        if (pip.IsEmpty() || login.IsEmpty() || password.IsEmpty() ||
confirmPassword.IsEmpty()) {
            ShowMessage("Будь ласка, заповніть всі обов'язкові поля .");
            return;
        }
        if (password != confirmPassword) {
            ShowMessage("Паролі не співпадають.");
            EditPas->Clear();
            EditPas2->Clear();
            EditPas->SetFocus();
            return;
        }
        if (password.Length() < 4) {
            ShowMessage("Пароль має бути не менше 4 символів.");
            return;
        }
        if (IsLoginExists(login)) {
            ShowMessage("Користувач з таким логіном вже існує.");
            EditLog->Clear();
            EditLog->SetFocus();
            return;
        }
        try
        {
            if (ADOQuery1->Active) {
                ADOQuery1->Close();
            }

            ADOQuery1->SQL->Clear();
            ADOQuery1->SQL->Add("INSERT INTO [User] (ПІП, Логін, Пароль, Роль)
VALUES (:PIP, :Login, :Password, :Role)");

            ADOQuery1->Parameters->ParamByName("PIP")->Value = pip;
            ADOQuery1->Parameters->ParamByName("Login")->Value = login;
            ADOQuery1->Parameters->ParamByName("Password")->Value = password;
            ADOQuery1->Parameters->ParamByName("Role")->Value = role;

            ADOQuery1->ExecSQL();

            ShowMessage("Реєстрація успішна! Ласкаво просимо, " + login + "!");

            ModalResult = mrOk;
        }
        catch (Exception &E)
        {
            ShowMessage("Помилка під час реєстрації: " + E.Message);
        }
        if (ADOQuery1->Active) {
            ADOQuery1->Close();
        }
    }
    //-----
void __fastcall TFormReg::Button2Click(TObject *Sender)
{
    FormAutor->Show();
    Hide();
}
    //-----

```

